DRS2016
Design + Research + Society
Future-Focused Thinking

2016 Design Research Society
50th Anniversary Conference
27-30 June 2016, Brighton, UK

The Idea of Architecture, The User As Inhabitant: Design through a Christopher Alexander Lens

Molly Wright Steenson

Carnegie Mellon University, School of Design
steenson@cmu.edu
DOI: 10.21606/drs.2016.127

Abstract: The architect Christopher Alexander contributed the major notion of "architecture" to the field of design research and its associated practices, an engagement that began in 1962, with the Design Methods movement. As co-author of *A Pattern Language* and author of such books as *Notes on the Synthesis of Form* and *The Timeless Way of Building,* he and his colleagues influenced the notion of architecture in the fields of design, design research and computer programming. Through the joint interpretations and applications of Alexander's version of architecture by designers and programmers, two important practices emerged: the use of patterns in programming, and the concept of a user as the inhabitant of a system of software. Yet this view of "architecture" held by designers and programmers differs from how architects practice the field, not to mention how they negatively they assess Alexander. Ultimately, there is much to learn from Alexander's contrarian stance and in the connections and disconnections in the idea of architecture, as it is understood in user-centered design.

Keywords: architecture, Christopher Alexander, pattern languages

Introduction

Designers and design researchers use the word "architecture" to describe the process of designing systems. In so doing, they envision the user of such a system as an inhabitant of the space of software. The architect Christopher Alexander, a seminal contributor to Design Methods and design research, is largely responsible for transmitting "architecture" to the practices of design researchers, user-centered designers (by which I include interaction, experience, and software designers, and information architects who engage with design research methods), and certain computer programmers.

In this paper, I explore this architectural transmission of Alexander's theories and practices to design research and interactive design practice, how it contributed to the idea of the user-as-inhabitant, and the ramifications of this transference. I will start with the way that first programmers, then later designers applied Alexander's patterns and his design politics in practice. (Programmers were the first to explicitly apply patterns to software design, which is why I include them in a paper oriented toward design research concerns.) Then, I will investigate how designers and programmers used "architecture" in this sense to develop the idea of the user as an inhabitant of a system, using the collection of essays Bringing Design to Software, edited by Terry Winograd as a specific set of examples. Finally, I will look at the way that these ideas differ from the concerns of traditional architectural practice. What can both design research and architecture learn from this divergence between the way system designers envision architecture and the way that architects practice it?

Christopher Alexander: A Brief Biography

From his undergraduate studies onward, Alexander developed a systematic approach to order in architecture through a combined interest in mathematics, computation and design. Born in 1936 in Vienna and raised in England, Alexander completed two bachelor's degrees at Trinity College at the University of Cambridge: the first in mathematics; the second in architecture. He continued his architectural studies in the doctoral program at Harvard starting in 1958, during which time he engaged in a variety of collaborations outside of architecture, including with Harvard's Center for Cognitive Research, the Joint Center for Urban Studies of MIT and Harvard, and the MIT Civil Engineering Systems Laboratory. These collaborations introduced him to cognitive science, cybernetics, and artificial intelligence, all of which informed the design methods he developed. He was one of the first and very few architects in the early 1960s with the mathematical expertise to program and use a computer, which led him to using a computer until the late '60s as a tool for modelling design problems.

Alexander was an important contributor to the Design Methods Movement, despite his eventual acrimonious criticism of it. He attended the 1962 Conference on Design Methods, and despite his absence from the 1967 Conference on Design Methods in Architecture, his influence was so central that Geoffrey Broadbent and Anthony Ward (1969) wrote in the conference proceedings, "Perhaps because Alexander is the only architect to have had such an influence in his own field, much of the symposium was devoted to a philosophical and operational analysis of his work" (pp. 7 & 9). In 1967, Alexander founded the Center for Environmental Structure at UC Berkeley. He and his colleagues shifted their focus away from computers and toward patterns and pattern languages, and published a number of books and other publications. Alexander left Berkeley in 1994 and between 2002–04, he published The Nature of Order: An Essay on the Art of Building and the Nature of the Universe, a four-volume theory of Alexandrian philosophy. He now lives in Surrey, England.

Alexander's work influences designers and programmers on a number of levels. At its core, it suggests methods for managing the information challenges of design and programming.

Notes on the Synthesis of Form (1964), the published version of his dissertation introduces the concept of fitness between form and context, as well as the relationship between the minutiae of a design problem and the greater sum of its whole. A Pattern Language, written by Alexander and his colleagues at the Center for Environmental Structure (1977), provides a modularized approach to design, suggesting to designers and programmers ways to capture and share the knowledge in a project for future use and evaluation. On a philosophical level, Alexander's theories undergird a politics of design that is local and place-based, flexible and negotiable, as he articulates in The Timeless Way of Building (1979), a book meant to accompany A Pattern Language. And finally, on a deeper, cosmological level, many designers and programmers find meaning in Alexander's appeals to a universal sense of order, as outlined in The Nature of Order (2002–04). The fact that his work operates on these different planes lends depth to the influence that it exercises on designers, design researchers and programmers. But it also means that different audiences take up his ideas in different ways.

Patterns, Programmers, and Politics

The Center for Environmental Structure honed the practice of pattern use over a period of a decade, starting in 1967 and culminating in A Pattern Language. A pattern solves "a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice," Alexander and co-authors Sara Ishikawa and Murray Silverstein (1977) wrote in A Pattern Language (p. x). A pattern evokes, represents, and describes an image such that it can be acted upon and built. "These patterns in our minds are, more or less, mental images of the pattern in the world: they are abstract representations of the very morphological rules which define the patterns in the world," Alexander (1979) wrote in The Timeless Way of Building (p. 181). The language of patterns is a format that organizes the parts, wholes and relationships in a design problem. It offered a structure for moving through the more intuitive shaping of the design problem, for applying it to a wide variety of problems, not just those of the traditional built environment.

Patterns and pattern languages have long proven attractive to audiences outside of architecture. Laypeople know his work, and A Pattern Language remains the #1 bestseller on Amazon's list of architectural criticism books. More specifically, Alexander's work attracted the attention of programmers and software designers starting in the late 1980s, who picked up his ideas and started applying them to software. A mimeographed copy of A Pattern Language allegedly passed hand-to-hand between software engineers, and some programmers had already encountered Alexander's books in their high school and college libraries and bookstores (K. Beck & A. Cooper, personal communication, 2015). Patterns proved attractive to programmers and early software designers because they incorporated the logic of computational processes, even though they didn't require a computer. Programmers and designers of systems struggled with the relationship of smaller parts to larger wholes, the intricacy of the fine details of code with the larger functions of programs and systems.

Programmers Kent Beck and Ward Cunningham were the first to apply Alexander's patterns in software: as they grappled with user interfaces in the Smalltalk object-oriented programming language, they applied patterns to the problem. In so doing, they found that patterns provided a useful way to distill and communicate the best solutions that they devised because patterns were concrete enough to capture the problem and abstract enough that other programmers could use them. Beck and Cunningham presented their work in a paper at the OOPSLA '87 conference (Object-Oriented Programs, Systems, Languages and Applications). Throughout the late '80s and early '90s, a growing community of engineers and programmers fleshed out patterns in programming languages, culminating in the best-selling book Design Patterns: Elements of Reusable Object-Oriented Software by the "Gang of Four" (Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides) in 1994. Today, there are hundreds if not thousands of books on patterns for software, hardware, the web, game design, and programming—a cursory Amazon.com search yields over 6500 titles that refer to patterns in a computational sense.

More than just a way to codify design and programming decisions, A Pattern Language and The Timeless Way of Building expressed the social and political human truths of the design process. In particular, these methods empowered the user of a system. From a programming perspective, Beck specifically saw patterns as a way to put in place the flexibility for a user to adapt to his or her needs. The concept of "gradual stiffening" in Pattern 208 in A Pattern Language (1977), influenced this idea: "The fundamental philosophy behind the use of pattern languages is that buildings should be uniquely adapted to individual needs and sites; and that the plans of buildings should be rather loose and fluid, in order to accommodate those subtleties" (p. 963). Gradual stiffening meant that the parts of a design could be moved around before they set. Not all aspects of design needed to be specified in advance on paper; instead, they could be tested in situ, shifted as needed, and firmed up in time.

Alexander's influence on the patterns in software movement was great enough that the object-oriented programming community invited Alexander to deliver the keynote at the OOPSLA '96 conference. In his talk, Alexander encouraged programmers to think beyond just using patterns as a programming tool. They were meant to be more than just a matter of capturing design decisions; the language and theory behind them had a higher meaning. By this point in his career, Alexander believed that this moral goodness was something that could be explicitly defined and empirically tested in architecture. He suggested that computer code might carry forward the genetic and linguistic frameworks for producing patterns. "I am convinced that the equivalent of the genes that act in organisms will have to be—or at least can be—software packages, acting in society" (Alexander, 1999, pp. 79–80). If programmers enacted it in this way, it could put decision-making in the hands of everyday individuals, much the way that Kent Beck interpreted Alexander's work as a tool for social and political democratization of the design process. Alexander (1999) asked to what extent programmers might be willing to take on "the responsibility for influencing, shaping, and changing the environment"(p. 81). If programmers and engineers were willing to take on patterns and their moral imperatives, then it was upon them to build a better world—not just a world imbued with computers, but with a deep understanding of how they contribute to the ecosystem in which they are located. "This is an extraordinary vision of the future, in which computers play a fundamental role in making the world—and above all the built structure of the world—alive, humane, ecologically profound, and with a deep living structure" (Alexander, 1999, p. 82).

Alexander's influence on programming practices continues today. Beck developed Extreme Programming (XP), a set of programming approaches that break down programming tasks into smaller, incrementally planned pieces and that asks programmers to code in pairs and continually test their code as they produce it. XP embodies values ("communication, feedback, simplicity, courage, and respect") that bind the community—values that are explicitly drawn from Alexander's work—Beck titled the final chapter in his book Extreme Programming Explained (2000) "The Timeless Way of Programming" (p. 2). Similarly, Alexander also inspired the wiki, the web platform that runs Wikipedia. In 1994, Cunningham developed an open-ended database of "people, products and patterns," calling it the WikiWikiWeb after the Hawaiian word for "quick;" he invited others to join, create and collaboratively edit it starting in March 1995. Cunningham never patented the wiki. The biggest user of the format, Wikipedia was founded in 2001 and has almost 5 million content pages, more than 794 million edits, more than ten edits per second and more than 26 million users as of October 2015, according to Wikipedia's Statistics page.

The User-as-Inhabitant

Through Alexander's architectural theories, digital designers began to understand the user as an inhabitant of a system of software. This approach suggested that instead of handing down plans and pronouncements, the designer of a system needed to think of the user as someone dwelling within its space. In particular, designers took up Alexander's concept of the "quality without a name," that Alexander introduces in The Timeless Way of Building, one that refers to a "central quality which is the root criterion of life and spirit in a man, a town, a building, or a wilderness" (p. ix). As a user became an inhabitant of a system, the question of software design became one that sought to accommodate these notions of "life."

In 1996, Stanford HCI Professor Terry Winograd published the edited volume Bringing Design to Software. The book had been in development since the early 1990s through a series of conversations and workshops, and its publication coincided with the beginning of the commercial explosion of the World Wide Web. Throughout the book, there are echoes both direct and indirect of Alexander. The authors throughout the book refer to patterns and languages in advocating for the user of software as an inhabitant of a system. In the introductory essay, Winograd wrote, Software is not just a device with which the user interacts; it is also the generator of a space in which the user lives. Software design is like architecture: When an architect designs a home or an office building, a structure is being specified. More significantly, though, the patterns of life for its inhabitants are being shaped. People are thought of as inhabitants rather than as users of buildings. In this book, we approach software users as inhabitants, focusing on how they live in the spaces that designers create. Our goal is to situate the work of the designer in the world of the user (1996, p. xvii).

I want to draw attention to two terms Winograd uses: inhabitant and generator. First, the inhabitant is not a mere functional user. It is a person whose life is shaped by the structures that surround her. By elevating the concerns of the user to that of an inhabitant, software design needed to address a deeper set of human needs. By invoking Alexander's "patterns of life," Winograd referred not just to patterns in a pattern language as a means of codifying information about design requirements: they instead represent truths about the experience of the user. Designing for an inhabitant of software insinuated a higher level of meaning—the "quality without a name" to which Alexander refers. It also suggested a different approach to design: if designers were to develop software that had a deeper quality to it, they would need to come to understand users in a new way. They would need to envision an inhabitant that lives in the space of software, one that generates possibilities for life.

Second, Winograd's statement of software as "the generator of a space" [emphasis mine] refers to a core concept for Alexander about systems, their propagation, and the production of the life quality that so interests him. Alexander wrote in his tidy 1968 article "Systems Generating Systems" that systems contain within them two parts: the "system as whole" that reflects the interaction of its parts, and a generating system, a "kit of parts, with rules about the way these parts may be combined," and that generate almost all "systems as a whole" (p. 605). The pattern language is a kit of parts and ruleset to govern it, but also "like a seed, is the genetic system which gives our millions of small acts the power to form a whole," Alexander wrote in The Timeless Way of Building (p. xiii). When Winograd introduced the idea of software being the "generator of a space in which the user lives," software became not only a mechanism that reproduces itself, but also a system that operates on a plane that affects life experience. As Alexander wrote in The Timeless Way of Building, "The patterns in the world merely exist. But the same patterns in our minds are dynamic. They have force. They are generative. They tell us what to do; they tell us how we shall, or may, generate them; and they tell us too, that under certain circumstances, we must create them" (p. 182).

The individual authors of Bringing Design to Software approach the user-as-inhabitant, interpreting "architecture" through Alexander's lens. In his "Software Design Manifesto," Mitch Kapor wrote about the "critical role of design, as a counterpart to programming," and suggested architecture as a place to look for inspiration for tools and practices (1996, p. 3). "In both architecture and software design it is necessary to provide the professional practitioner with a way to model the final result with far less effort than is required to build the final product. In each case, specialized tools and techniques are used," Kapor wrote, tools that software design did not yet have (p. 6). Architecture's focus on models could provide means for designers to bridge the abstract and the conceptual, the underlying structure and the overarching experience. Kapor also suggested circumspection about the roles of the architect vis-à-vis that of the engineer. "Architecture and engineering are, as disciplines, peers to each other, but in the actual process of designing and implementing the building, the engineers take direction from the architects. The engineers play a vital and crucial role in the process, but they take their essential direction from the design of the building as established by the architect" (p. 4). So, too, it should be in the design of software: the new role of the software designer should interpret the needs of the inhabitant, bridging the abstract and the concrete, and deliver models for the engineer to implement. Yet the pattern-oriented software engineers such as Kent Beck, Ward Cunningham, and the "Gang of Four" who introduced patterns to the object-oriented community would likely have objected to this very handoff. They might have argued that engineers could take architecture into their hands—especially architecture as derived from Alexander's work. Patterns could provide a bridging role and could offer a means for engineers to consider inhabitants, too.

In their essay "Design Languages," John Rheinfrank and Shelley Evenson took up Alexander's notion of language, applying it to design in the space of software. They advocated using design languages to translate complex actions into simpler steps, and providing pathways for mastering unfamiliar tasks—whether the repair of a photocopier, or the navigation of new software interfaces. "Design languages are present everywhere in our constructed environment. Most design languages have evolved through unconscious design activities," they wrote (1996, p. 65). Similar to how Alexander constituted language as an organizing principle for patterns, Rheinfrank and Evenson defined design languages as a universal way to relate the parts of a problem into a greater whole:

Just as natural (spoken or written) languages are the basis for how we generate and interpret phrases and sentences, so design languages are the basis for how we create and interact with things in the world. And, like spoken or written language, design languages are assimilated into our everyday activities, mediating our experiences with the world (often tacitly), and contributing to the perceived quality of our lives (p. 65).

The development of design languages that Rheinfrank and Evenson advocated follows a five-step process: characterization (setting out assumptions so that they could be challenged or "recast"), reregistration (developing a new framework), development and registration (making concrete the framework through scenarios and specifications), evaluation (seeing the language in context) and evolution (how it develops beyond its current needs—and considering the cycle anew). Through a focus on context in use, ever-evolving design languages benefit the user-as-inhabitant, providing a mode for understanding and for communication.

Differing and Disconnecting Architectures?

Yet these interpretations of designing for the user-as-habitant differed from the concerns of traditional, practicing architects. In examining the ways that designers and programmers interpret Alexander's notion of architecture in support of the idea of an inhabitant of a system, it is also important to see what they leave out of their perspective. Bringing Design to Software addressed this split in the "Software Design and Architecture" chapter by Winograd and the architect Philip Tabor, who warned against embracing architecture uncritically.[1] "We in the software profession may have much to learn from the ancient and rich tradition of architectural practice and architectural theory," they wrote. "At the same time, in drawing such a broad analogy, it is possible to fall into superficiality, finding attractive but misleading guidance" (Winograd & Tabor, 1996, p. 10). For one, what is at stake for architects may not be the same as for their users. Architects don't typically work for end users, they work for clients. For another, the division of labor deserves to be questioned. Engineers determine the structure, architects address a program and determine its look, feel, and spatial experience, and builders construct the building. The relationship begged other questions: what about the legal liability that architects, engineers and developers hold in traditional design and build: should software architects and designers have this same responsibility?

Winograd was not quite correct about architecture not having users, and paradoxically, this suggests a possible bridge for these disparate perspectives on architecture. In his recent edited volume Use Matters: An Alternative History of Architecture," Kenny Cuppers writes, "Utility is central to what architects do in practice as they deal with clients, norms, and building regulations… But utility also governs an unknowable universe of everyday experience that remains outside of the designer's direct control" (2013, p. 1). The everyday practice of architecture, he argues, is not an exercise of the primacy of form—it is a negotiation between drawing, computation, bureaucracy, and communication. "If a lot of architecture's meaning is made not on the drafting board but in the complex lifeworld of how it is inhabited, consumed, used, lived or neglected, that world is at once central and peculiarly under-explored" (p. 1). Architectural practice could learn from the way that software designers, researchers and programmers have taken up use and utility, and the ways that for decades, their practices have postulated an inhabitant of the space of software. What might happen if architects discover this new kind of user?

Finally, the notion of architecture as interpreted through the lens of Alexander differed—and still does differ—from the concerns of most architects who design the form of a building, one that they define through a set of representations and models (whether sketches and drawings, CAD and parametric files and renderings, or cardboard, wood, 3D printed or milled models). Alexander's method defines the structure of a design problem by

[1] Philip Tabor has been on the faculty of the Bartlett (a school of architecture at University College London) for over three decades and also taught at the Interaction Design Institute Ivrea in Ivrea, Italy, a school founded by his wife Gillian Crampton Smith (with whom he contributed another essay in the volume). Crampton Smith founded the Computer-Related Design program at the Royal College of Art and was Director of the Interaction Design Institute Ivrea in Ivrea, Italy.

representing its parts in an information system flexible enough to adapt over time. He was and is disinterested, to say the least, in novel form and representation, far preferring vernacular architecture and Turkish carpets, seeking out the systemic order underlying them, and deriving from them an objectively verifiable moral good.[1] Where designers, programmers, and laypeople find useful tools, an articulation of design politics, and even a cosmological sense of truth, architects find themselves at odds, to put it mildly, with Alexander's approach: they find his process reductive, and his tone moralizing, as a number of architects stated in a discussion on a Facebook post I started about the different ways that architects, designers and technologists view Alexander (personal communication, January 8, 2015). Architects are sometimes suspicious of Alexander's lay popularity and reject his work. Dominant trends in architectural discourse, contemporaneous with Alexander's work and continuing today, support architecture's autonomy and underscore form and representation. Alexander's work, especially in the last 15 years, seems universes away from these concerns. Moreover, architects guard the title of "architect": they can't even call themselves architects until they pass an extensive set of licensing exams. Consider one example: Nathan Shedroff, an early information architect and founder of Vivid Studios, an early web consulting and information architecture firm, received a letter from the California Board of Architectural Examiners in 1999 in response to a column he wrote in New Media magazine titled "THE ARCHITECT," demanding he not use that title (Suarez). Indeed, architects don't use the word "architect" as a verb—professional architects "design." Technically-oriented designers and engineers "architect" complex systems, using a verb form that architects do not use.

In these apparent disconnects, there are productive tensions. What might architects learn from the idea of systems of the user-as-inhabitant? What might designers learn from forays into the radical forms of parametric architecture? Might there be new design processes that could emerge from a new set of patterns that could emerge? What could we all learn from the ethical and moral questions that Alexander asked his keynote audience about the propagation of computer systems? In reality, we are all inhabitants of the space of software—it surrounds us, we live within it. Alexander's work played a major role in that conception.

Conclusion

In this paper, I delved into the contribution of the notion of architecture to software design, design research and programming through the work of Christopher Alexander. Through the joint interpretations and applications of his work by programmers and designers, Alexander's work brought to bear the concept of a new kind of user: the inhabitant of a

[1] For example, Alexander's critique of a particular building that he writes about in *The Nature of Order:* "Compare the pretentious plastic-fantastic postmodern 'house' on this page. It is a horrible deathly thing. Under normal circumstances this would not even be worth commenting on. But things have become so topsy-turvy in our world that this building is considered a valid work of architecture, worthy of being illustrated in architectural magazines, while the slum above is considered terrible... while the postmodern house with its image-ridden knobs and ears perhaps has little to do with life, little to do with any deep reality" (2002, p. 59).

system of software. Designing for an inhabitant begged a new set of questions for designers, who used Alexander's concepts of patterns, generativity, and life to develop new user-centered design practices that could evolve over time, taking into consideration the fact that experience changes the spaces that we inhabit. Yet the metaphor of architecture that these designers advocated is not always true to the way that architects practice their craft. Despite his lay popularity and his uptake with designers, design researchers, and programmers, Alexander is a fringe character for the current practicing architectural mainstream and architectural academia—even if architects concede that he's an important figure. They just would rather not engage with him because of his moralizing arguments and what they argue is a reductiveness of his approach to architecture.

Yet there is much to learn from Alexander's contrarian approach, as well as from the disconnect between Alexander and the architects who dislike him, and a possibility to rehabilitate him. Moreover, in this digitally-imbricated world in which we dwell in smart cities and are surrounded by the Internet of Things, we all inhabit the space of software.

References

Alexander, C. (1968). Systems Generating Systems, *AD*, *38*, pp. 605–10.

Alexander, C. (2002). *The Nature of Order : An Essay on the Art of Building and the Nature of the Universe.* Berkeley, CA: Center for Environmental Structure.

Alexander, C. (1999). The Origins of Pattern Theory: The Future of the Theory, and the Generation of a Living World. *IEEE Software*, *16* (5), pp. 71–82.

Alexander, C. (1979). *The Timeless Way of Building*. New York: Oxford University Press.

Alexander, C., Ishikawa, S., & Silverstein, M. (1977) *A Pattern Language: Towns, Buildings, Construction*. New York: Oxford University Press.

Beck, K. (2000). *Extreme Programming eXplained: Embrace Change.* Reading, MA: Addison-Wesley.

Broadbent, G. & Ward, A. (Eds.). (1969). *Design Methods in Architecture.* London: Lund Humphries.

Cuppers, K. (2013). *Use Matters: An Alternative History of Architecture.* London: Routledge.

Kapor, M. (1996). A Software Design Manifesto. In T. Winograd (Ed.) *Bringing Design to Software* (pp. 1–9). New York: ACM Press.

Keller, S. (2005). *Systems Aesthetics: Architectural Theory at the University of Cambridge, 1960–75*. (Doctoral dissertation). Retrieved from ProQuest Dissertations and Theses (Proquest Document ID 305007831).

Rheinfrank, J. & Evenson, S. (1996). Design Languages. In T. Winograd (Ed.) *Bringing Design to Software* (pp. 63–80). New York: ACM Press.

Suarez, Chantel. Letter to Nathan Shedroff from Board of Architectural Examiners, State of California. 10 June 1999. TS.

Winograd, T. (Ed.) (1996). *Bringing Design to Software*. New York: ACM Press.

Winograd, T. & Tabor, P. Profile 1. Software Design and Architecture n T. Winograd (Ed.) *Bringing Design to Software* (pp. 10–16). New York: ACM Press.

About the Author:

Dr. Molly Wright Steenson is an associate professor at Carnegie Mellon University's School of Design. She holds a PhD from Princeton University in architecture and is finishing a book titled *Architecting Interactivity* about Christopher Alexander, Nicholas Negroponte, Richard Saul Wurman, and Cedric Price.